



**WELCOME**

**Domain Science & Engineering**  
**A Precursor for Requirements Engineering**

FM 2012 Tutorial, CNAM, 28 August 2012

**Dines Bjørner**

**DTU Informatics**  
**July 31, 2012: 09:02**

# **Begin of Lecture 1: First Session — Introduction**

## **Domains, TripTych, Issues**

**FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012**

# Tutorial Schedule

● Lectures 1–2	9:00–9:40 + 9:50–10:30	
✓ 1 <b>Introduction</b>		<b>Slides 1–35</b>
2 <b>Endurant Entities: Parts</b>		Slides 36–110
● Lectures 3–5	11:00–11:15 + 11:20–11:45 + 11:50–12:30	
3 <b>Endurant Entities: Materials, States</b>		Slides 111–142
4 <b>Perdurant Entities: Actions and Events</b>		Slides 143–174
5 <b>Perdurant Entities: Behaviours</b>		Slides 175–285
<b>Lunch</b>	<b>12:30–14:00</b>	
● Lectures 6–7	14:00–14:40 + 14:50–15:30	
6 <b>A Calculus: Analysers, Parts and Materials</b>		Slides 286–339
7 <b>A Calculus: Function Signatures and Laws</b>		Slides 340–377
● Lectures 8–9	16:00–16:40 + 16:50–17:30	
8 <b>Domain and Interface Requirements</b>		Slides 378–424
9 <b>Conclusion: Comparison to Other Work</b>		Slides 428–460
<b>Conclusion: What Have We Achieved</b>		Slides 425–427 + 461–472

## Summary

- This tutorial covers
  - ❖ a **new science & engineering of domains** as well as
  - ❖ a **new foundation for software development**

We treat the latter first.

- Instead of commencing with **requirements engineering**,
  - ❖ whose pursuit may involve repeated,
  - ❖ but unstructured forms of **domain analysis**,
  - ❖ we propose a predecessor phase of **domain engineering**.
- That is, we single out **domain analysis** as an activity to be pursued prior to **requirements engineering**.

- 
- In emphasising **domain engineering** as a predecessor phase
    - ❖ we, at the same time, introduce a number of facets
    - ❖ that are **not present**, we think,
    - ❖ in current software engineering studies and practices.
  - **One facet** is **the construction of separate domain descriptions.**
    - ❖ Domain descriptions are void of any reference to requirements
    - ❖ and encompass the **modelling of domain phenomena**
    - ❖ without regard to their being computable.

- **Another facet** is **the pursuit of domain descriptions as a free-standing activity.**
  - ❖ In this tutorial we emphasize domain description development need not lead to [further] software development.
  - ❖ This gives a new meaning to **business process engineering**, and should lead to
    - ⊗ a deeper understanding of a domain
    - ⊗ and to possible non-IT related **business process re-engineering** of areas of that domain.
- In this tutorial we shall investigate
  - ❖ a method for analysing domains,
  - ❖ for constructing domain descriptions
  - ❖ and some emerging scientific bases.

- **Our contribution** to **domain analysis** is that we view domains as having the following **ontology**.
  - ❖ There are the **entities** that we can describe and then there is “the rest” which we leave un-described.
  - ❖ We **analyse** entities into
    - ⊗ **endurant entities** (Slides 51–134) and
    - ⊗ **perdurant entities** (Slides 144–280).that is
    - ⊗ **parts and materials as endurant entities** (Slides 51–134) and
    - ⊗ **discrete actions, discrete events and behaviours as perdurant entities** (Slides 146–280), respectively.
- Another way of looking at **entities** is as
  - ❖ **discrete entities** (Slides 51–174 and 176–245),
  - ❖ **continuous entities** (Slides 112–134 and Slides 246–280).



- We also contribute to the **analysis of discrete endurants** in terms of the following notions:
  - ❖ **part types** and **material types** (Slides 51–69 and Slides 112–134),
  - ❖ **part unique identifiers** (Slides 75–77),
  - ❖ **part mereology** (Slides 78–93) and
  - ❖ **part attributes** and **material attributes** (Slides 94–105, Slides 121–125) and
  - ❖ **material laws** (Slides 126–131).
- Of the above we point to the introduction, into **computing science** and **software engineering** of the notions of
  - ❖ **materials** (Slides 112–134) and
  - ❖ **continuous behaviours** (Slides 246–280)

**as novel.**

---

## 1. Introduction

- I remind You of the **abstract**,
  - ❖ Slide 7,
  - ❖ as for the **contributions** of this tutorial.
- This is primarily a **methodology** paper.
- By a **method** we shall understand
  - ❖ a set of **principles**
  - ❖ for **selecting** and **applying**
  - ❖ a number of **techniques** and **tools**
  - ❖ in order to **analyse** a **problem**
  - ❖ and **construct** an **artefact**.
- By **methodology** we shall understand
  - ❖ the study and knowledge about methods.


- This tutorial contributes to
  - ❖ the study and knowledge
  - ❖ of software engineering development methods.
- Its contributions lie in by suggesting and exploring
  - ❖ **domain engineering** and
  - ❖ domain engineering as a basis for **requirements engineering**.
- We are not saying
  - ❖ “*thou must develop software this way*”,
- but we do suggest
  - ❖ that since it is possible
  - ❖ and makes sense to do so
  - ❖ it may also be wise to do so.

## 1.1. Domains: Some Definitions

- By a **domain** we shall here understand
  - ⋄ an area of human activity
  - ⋄ characterised by observable phenomena:
    - ⊗ **entities**
      - \* whether **endurants** (manifest **parts** and **materials**)
      - \* or **perdurants** (**actions**, **events** or **behaviours**),
    - ⊗ whether
      - \* **discrete**
      - \* **continuous**;
    - ⊗ and of their **properties**;

**Example: 1**   **Some Domains.**   Some examples are:

air traffic,  
airport,  
banking,  
consumer market,  
container lines,  
fish industry,  
health care,

logistics,  
manufacturing,  
pipelines,  
securities trading,  
transportation  
etcetera. 

## 1.1.1. Domain Analysis

- By domain analysis we shall understand
  - ⋄ an inquiry into the domain,
  - ⋄ its entities
    - ⊗ (manifest parts and materials;
    - ⊗ actions, events and behaviours)
  - ⋄ and their properties

## Example: 2 A Container Line Analysis.

We omit enumerating entity properties.

- *parts*:
  - ◇ container,
  - ◇ vessel,
  - ◇ terminal port, etc.;
- *actions*:
  - ◇ container loading,
  - ◇ container unloading,
  - ◇ vessel arrival in port, etc.;
- *events*:
  - ◇ container falling overboard;
  - ◇ container afire;
  - ◇ etc.;
- *behaviour*:
  - ◇ vessel voyage,
  - ◇ across the seas,
  - ◇ visiting ports, etc.

Length of a container is a container *property*.

Name of a vessel is a vessel *property*.

Location of a container terminal port is a port *property*.

## 1.1.2. Domain Descriptions

- By a domain description we shall understand
  - ❖ a narrative description
  - ❖ tightly coupled (say line-number-by-line-number)
  - ❖ to a formal description.
- To develop a domain description requires a thorough amount of **domain analysis**.



## Example: 3 A Transport Domain Description.

- *Narrative:*

- ◇ a transport net,  $n:N$ ,  
consists of an aggregation of hubs,  $hs:HS$ ,  
which we “concretise” as a set of hubs, **H-set**, and  
a set of links,  $ls:LS$ , respectively **L-set**,

- *Formalisation:*

**type**  $N, HS, LS, Hs = \text{H-set}, Ls = \text{L-set}, H, L$   
**value**

$obs\_HS: N \rightarrow HS,$   
 $obs\_LS: N \rightarrow LS.$   
 $obs\_Hs: N \rightarrow \text{H-set},$   
 $obs\_Ls: N \rightarrow \text{L-set}.$



### 1.1.3. Domain Engineering

- By domain engineering we shall understand
  - ⋄ the engineering of a domain description,
  - ⋄ that is,
    - ⊗ the rigorous construction of domain descriptions, and
    - ⊗ the further analysis of these, creating theories of domains<sup>1</sup>, etc.

---

<sup>1</sup>Examples of such theories, albeit in rather rough forms, are given in Appendices B–C.

- The size<sup>2</sup>, structure<sup>3</sup> and complexity<sup>4</sup> of interesting domain descriptions is usually such as to put a special emphasis on engineering:
  - ❖ the management and organisation of several, typically 5–6 collaborating domain describers,
  - ❖ the ongoing check of description quality, completeness and consistency, etcetera.

---

<sup>2</sup>usually, say a hundred pages

<sup>3</sup>usually a finely sectioned document of many subsub... subsections

<sup>4</sup>having many cross-references between subsub... subsections

## 1.1.4. Domain Science

- By domain science we shall understand
  - ◇ two things:
    - ⊗ the general study and knowledge of
      - \* how to create and handle domain descriptions
      - \* (a general theory of domain descriptions)
    - and
    - ⊗ the specific study and knowledge of a particular domain.
  - ◇ The two studies intertwine.

## 1.2. The Triptych of Software Development

- We suggest a “dogma”:
  - ❖ before **software** can be **designed**  
one must understand<sup>5</sup> the **requirements**; and
  - ❖ before **requirements** can be expressed  
one must understand<sup>6</sup> the **domain**.
- We can therefore view software development as ideally proceeding in three (i.e., **TripTych**) phases:
  - ❖ an initial phase of **domain engineering**, followed by
  - ❖ a phase of **requirements engineering**, ended by
  - ❖ a phase of **software design**.

---

<sup>5</sup>Or maybe just: have a reasonably firm grasp of

<sup>6</sup>See previous footnote!

- In the **domain engineering phase** ( $\mathcal{D}$ )
  - ❖ a domain is analysed, described and “theorised”,
  - ❖ that is, the beginnings of a **specific domain theory** is established.
- In the **requirements engineering phase** ( $\mathcal{R}$ )
  - ❖ a **requirements prescription** is constructed —
  - ❖ significant fragments of which are “derived”,
  - ❖ systematically, from the **domain description**.
- In the **software design phase** ( $\mathcal{S}$ )
  - ❖ a **software design**
  - ❖ is derived, systematically, rigorously or formally,
  - ❖ from the **requirements prescription**.
- Finally the **Software** is proven correct with respect to the **Requirements** under assumption of the **Domain**:  $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ .

- By a **machine** we shall understand the **hardware** and **software** of a target, i.e., a required **IT system**.
- In [dines:ugo65:2008,psi2009,Kiev:2010ptI] we indicate how one can “derive” significant parts of requirements from a suitably comprehensive domain description – basically as follows.
  - ❖ **Domain projection**: from a domain description one **projects** those areas that are to be somehow manifested in the software.
  - ❖ **Domain initialisation**: for that resulting projected requirements prescription one **initialises** a number of part types as well as action and behaviour definitions, from less **abstract** to more **concrete**, specific types, respectively definitions.

- ❖ **Domain determination:** hand-in-hand with domain initialisation a[n interleaved] stage of making values of types less **non-deterministic**, i.e., more **deterministic**, can take place.
- ❖ **Domain extension:** Requirements often arise in the context of new business processes or technologies either placing old or replacing human processes in the domain. Domain extension is now the ‘enrichment’ of the domain requirements, so far developed, with the description of these new business processes or technologies.
- ❖ Etcetera.
- The result of this part of “requirements derivation” is the **domain requirements**.



- A set of domain-to-requirements operators similarly exists for constructing **interface requirements**
  - ❖ from the domain description and,
  - ❖ independently, also from knowledge of the **machine**
  - ❖ for which the required IT system is to be developed.
- We illustrate the **techniques of domain requirements and interface requirements** in Sect. 4.
- Finally **machine requirements** are “derived”
  - ❖ from just the knowledge of the **machine**,
  - ❖ that is,
    - ⊗ the target hardware and
    - ⊗ the software system tools for that hardware.

- When you review this section  
(‘A Triptych of Software Development’)
  - ❖ then you will observe how ‘the domain’
  - ❖ predicates both the requirements
  - ❖ and the software design.
- For a specific domain one may develop
  - ❖ many (thus related) requirements
  - ❖ and from each such (set of) requirements
  - ❖ one may develop many software designs.
- We may characterise this multitude of domain-predicated requirements and designs as a **product line** [dines-maurer].
- You may also characterise domain-specific developments as representing another ‘definition’ of **domain engineering**.

## 1.3. Issues of Domain Science & Engineering

- We specifically focus on the following issues of domain science &<sup>7</sup> engineering:
  - ❖ (i) which are the “things” to be described<sup>8</sup>,
  - ❖ (ii) how to analyse these “things” into description structures<sup>9</sup>,
  - ❖ (iii) how to describe these “things” informally and formally,
  - ❖ (iv) how to further structure descriptions<sup>10</sup>, and a further study of
  - ❖ (v) mereology<sup>11</sup>.

---

<sup>7</sup>When we put ‘&’ between two terms that the compound term forms a whole concept.

<sup>8</sup>endurants [manifest entities henceforth called **parts** and **materials**] and perdurants [actions, events, behaviours]

<sup>9</sup>atomic and composite, unique identifiers, mereology, attributes

<sup>10</sup>*intrinsic, support technology, rules & regulations, organisation & management, human behaviour etc.*

<sup>11</sup>the study and knowledge of parts and relations of parts to other parts and a “whole”.

## 1.4. Structure of Paper

- It is always a good idea to consult and study the *table of contents* listing of the colloquium one is listening to. Therefore one is brought here:

<b>Introduction</b>	<b>9</b>
<b>Domains: Some Definitions</b>	<b>11</b>
<b>Domain Analysis</b>	<b>13</b>
<b>Domain Descriptions</b>	<b>15</b>
<b>Domain Engineering</b>	<b>17</b>
<b>Domain Science</b>	<b>19</b>
<b>The Triptych of Software Development</b>	<b>20</b>
<b>Issues of Domain Science &amp; Engineering</b>	<b>26</b>
<b>Structure of Paper</b>	<b>27</b>

---

<b>Domain Entities</b>	<b>37</b>
<b>From Observations to Abstractions</b> . . . . .	39
<b>Algebras</b> . . . . .	40
<b>Phenomena</b> . . . . .	41
<b>Entities</b> . . . . .	42
<b>A Description Bias</b> . . . . .	43
<b>An ‘Upper Ontology’</b> . . . . .	45

<b>Endurants</b>	<b>48</b>
<b>General</b>	49
<b>Discrete and Continuous Endurants</b>	49
<b>Discrete Endurants: Parts</b>	<b>51</b>
<b>Atomic and Composite Parts</b>	51
<b>Atomic Parts</b>	54
<b>Composite Parts</b>	63
<b>Abstract Types, Sorts, and Concrete Types</b>	65
<b>Properties</b>	70
<b>Unique Identification</b>	75
<b>Mereology</b>	78
<b>Attributes</b>	93
<b>Shared Attributes and Properties</b>	100
<b>Attribute Naming</b>	100
<b>Attribute Sharing</b>	103
<b>Shared Properties</b>	104
<b>Summary of Discrete Endurants</b>	105

---

<b>Continuous Endurants: Materials</b>	<b>111</b>
<b>“Somehow Related” Parts and Materials</b> . . . . .	113
<b>Material Observers</b> . . . . .	115
<b>Material Properties</b> . . . . .	120
<b>Material Laws of Flows and Leaks</b> . . . . .	125
<b>A Final Note on Endurant Properties</b>	<b>132</b>
<b>States</b>	<b>134</b>
<b>General</b> . . . . .	134
<b>State Invariants</b> . . . . .	137

---

<b>Discrete Perdurants</b>	<b>143</b>
<b>General</b>	143
<b>Discrete Actions</b>	145
<b>An Aside on Actions</b>	147
<b>Action Signatures</b>	149
<b>Action Definitions</b>	150
<b>Discrete Events</b>	161
<b>An Aside on Events</b>	163
<b>Event Signatures</b>	164
<b>Event Definitions</b>	165



---

<b>Discrete Behaviours</b> . . . . .	175
<b>What is Meant by ‘Behaviour’ ?</b> . . . . .	177
<b>Behaviour Narratives</b> . . . . .	180
<b>An Aside on Agents, Behaviours and Processes</b> . . . . .	184
<b>On Behaviour Description Components</b> . . . . .	186
<b>A Model of Parts and Behaviours</b> . . . . .	206
<b>Sharing Properties <math>\equiv</math> Mutual Mereologies</b> . . . . .	215
<b>Behaviour Signatures</b> . . . . .	217
<b>Behaviour Definitions</b> . . . . .	220

---

<b>Continuous Perdurants</b>	<b>236</b>
<b>Some Examples</b> . . . . .	237
<b>Motivation for Consolidated Models</b> . . . . .	242
<b>Generation of Consolidated Models</b> . . . . .	246
<b>The Pairing Process</b> . . . . .	247
<b>Matching</b> . . . . .	249
<b>An Aside on Time</b> . . . . .	250
<b>A Research Agenda</b> . . . . .	251
<b>Discussion</b> . . . . .	252
 <b>Discussion of Entities</b>	 <b>253</b>

<b>Towards a Calculus of Domain Discoverers</b>	<b>258</b>
<b>Introductory Notions</b>	261
<b>Discovery</b>	261
<b>Analysis</b>	264
<b>Domain Indexes</b>	265
<b>The Repository</b>	271
<b>Domain Analysers</b>	274
IS_MATERIALS_BASED	275
IS_ATOM, IS_COMPOSITE	277
HAS_A_CONCRETE_TYPE	281

<b>Domain Discoverers</b> . . . . .	283
MATERIAL_SORTS . . . . .	284
PART_SORTS . . . . .	286
PART_TYPES . . . . .	290
UNIQUE_ID . . . . .	293
MEREOLGY . . . . .	294
ATTRIBUTES . . . . .	298
ACTION_SIGNATURES . . . . .	306
EVENT_SIGNATURES . . . . .	310
BEHAVIOUR_SIGNATURES . . . . .	313
<b>Order of Analysis and “Discovery”</b> . . . . .	320
<b>Analysis and “Discovery” of “Leftovers”</b> . . . . .	321

---

<b>Laws of Domain Descriptions</b> . . . . .	323
<b>1st Law of Commutativity</b> . . . . .	325
<b>2nd Law of Commutativity</b> . . . . .	327
<b>3rd Law of Commutativity</b> . . . . .	329
<b>1st Law of Stability</b> . . . . .	331
<b>2nd Law of Stability</b> . . . . .	332
<b>Law of Non-interference</b> . . . . .	333
<b>Discussion</b> . . . . .	336

---

<b>Requirements Engineering</b>	<b>342</b>
<b>The Transport Domain — a Resumé</b>	343
<b>Nets, Hubs and Links</b>	343
<b>Mereology</b>	344
<b>A Requirements “Derivation”</b>	346
<b>Definition of Requirements</b>	346
<b>The Machine = Hardware + Software</b>	347
<b>Requirements Prescription</b>	348
<b>Some Requirements Principles</b>	349
<b>A Decomposition of Requirements Prescription</b>	350
<b>An Aside on Our Example</b>	351

---

<b>Domain Requirements</b> . . . . .	353
<b>Projection</b> . . . . .	357
<b>Instantiation</b> . . . . .	358
<b>Determination</b> . . . . .	363
<b>Extension</b> . . . . .	367
<b>Interface Requirements Prescription</b> . . . . .	372
<b>Shared Parts</b> . . . . .	374
<b>Shared Actions</b> . . . . .	380
<b>Shared Events</b> . . . . .	381
<b>Shared Behaviours</b> . . . . .	382
<b>Machine Requirements</b> . . . . .	383
<b>Discussion of Requirements “Derivation”</b> . . . . .	384

<b>Conclusion</b>	<b>388</b>
<b>Comparison to Other Work</b>	<b>390</b>
<b>Ontological Engineering:</b>	<b>391</b>
<b>Knowledge and Knowledge Engineering:</b>	<b>394</b>
<b>Domain Analysis:</b>	<b>398</b>
<b>Software Product Line Engineering:</b>	<b>400</b>
<b>Problem Frames:</b>	<b>402</b>
<b>Domain Specific Software Architectures (DSSA):</b>	<b>405</b>
<b>Domain Driven Design (DDD)</b>	<b>410</b>
<b>Feature-oriented Domain Analysis (FODA):</b>	<b>412</b>
<b>Unified Modelling Language (UML)</b>	<b>414</b>
<b>Requirements Engineering:</b>	<b>417</b>
<b>Summary of Comparisons</b>	<b>420</b>
<b>What Have We Achieved and Future Work</b>	<b>421</b>
<b>General Remarks</b>	<b>424</b>
<b>Acknowledgements</b>	<b>431</b>



<b>On A Theory of Transport Nets</b>	<b>433</b>
<b>Some Pictures</b>	434
<b>Parts</b>	440
<b>Nets, Hubs and Links</b>	440
<b>Mereology</b>	441
<b>An Auxiliary Function</b>	442
<b>Retrieving Hubs and Links</b>	443
<b>Invariants over Link and Hub States and State Spaces</b>	444
<b>Maps</b>	448
<b>Routes</b>	452
<b>Special Routes</b>	455
<b>Special Maps</b>	457
<b>Actions</b>	459
<b>Insert Hub</b>	459
<b>Insert Link</b>	461
<b>Remove Hub</b>	464
<b>Remove Link</b>	465

<b>On A Theory of Container Stowage</b>	<b>466</b>
<b>Some Pictures</b> . . . . .	467
<b>Parts</b> . . . . .	470
<b>A Basis</b> . . . . .	470
<b>Mereological Constraints</b> . . . . .	477
<b>Stack Indexes</b> . . . . .	478
<b>Stowage Schemas</b> . . . . .	487
<b>Actions</b> . . . . .	492
<b>Remove Container from Vessel</b> . . . . .	492
<b>Remove Container from CTP</b> . . . . .	495
<b>Stack Container on Vessel</b> . . . . .	496
<b>Stack Container in CTP</b> . . . . .	497
<b>Transfer Container from Vessel to CTP</b> . . . . .	498
<b>Transfer Container from CTP to Vessel</b> . . . . .	499

<b>RSL: The Raise Specification Language</b>	<b>500</b>
<b>Type Expressions</b>	500
Atomic Types	500
Composite Types	502
<b>Type Definitions</b>	505
Concrete Types	505
Subtypes	507
Sorts — Abstract Types	508
<b>The RSL Predicate Calculus</b>	509
Propositional Expressions	509
Simple Predicate Expressions	510
Quantified Expressions	511

<b>Concrete RSL Types: Values and Operations</b> . . . . .	512
<b>Arithmetic</b> . . . . .	512
<b>Set Expressions</b> . . . . .	513
<b>Cartesian Expressions</b> . . . . .	515
<b>List Expressions</b> . . . . .	516
<b>Map Expressions</b> . . . . .	518
<b>Set Operations</b> . . . . .	520
<b>Cartesian Operations</b> . . . . .	525
<b>List Operations</b> . . . . .	526
<b>Map Operations</b> . . . . .	532
<b><math>\lambda</math>-Calculus + Functions</b> . . . . .	537
<b>The <math>\lambda</math>-Calculus Syntax</b> . . . . .	537
<b>Free and Bound Variables</b> . . . . .	538
<b>Substitution</b> . . . . .	539
<b><math>\alpha</math>-Renaming and <math>\beta</math>-Reduction</b> . . . . .	540
<b>Function Signatures</b> . . . . .	541
<b>Function Definitions</b> . . . . .	542

<b>Other Applicative Expressions</b> . . . . .	544
<b>Simple let Expressions</b> . . . . .	544
<b>Recursive let Expressions</b> . . . . .	545
<b>Predicative let Expressions</b> . . . . .	546
<b>Pattern and “Wild Card” let Expressions</b> . . . . .	547
<b>Conditionals</b> . . . . .	548
<b>Operator/Operand Expressions</b> . . . . .	549
<b>Imperative Constructs</b> . . . . .	550
<b>Statements and State Changes</b> . . . . .	550
<b>Variables and Assignment</b> . . . . .	551
<b>Statement Sequences and skip</b> . . . . .	552
<b>Imperative Conditionals</b> . . . . .	553
<b>Iterative Conditionals</b> . . . . .	554
<b>Iterative Sequencing</b> . . . . .	555

---

<b>Process Constructs</b> . . . . .	556
<b>Process Channels</b> . . . . .	556
<b>Process Composition</b> . . . . .	557
<b>Input/Output Events</b> . . . . .	558
<b>Process Definitions</b> . . . . .	559
<b>Simple RSL Specifications</b> . . . . .	560

- First (Sect. 1) we introduce the problem. And that was done above.

- Then, in (Sects. 2–10)
  - ❖ we bring a rather careful analysis of
  - ❖ the concept of the **observable, manifest phenomena**
  - ❖ that we shall refer to as **entities**
- We strongly think that these sections of this tutorial
  - ❖ brings a simple, to our taste, simple and elegant
  - ❖ reformulation of what is usually called “*data modelling*”,
  - ❖ in this case for domains —
  - ❖ but with major aspects applicable as well to
  - ❖ **requirements development and software design.**

- That analysis focuses on
  - ⋄ **endurant entities**, also called **parts** and **materials**,
    - ⊗ those that can be observed at no matter what time,
    - ⊗ i.e., entities of substance or continuant, and
  - ⋄ **perdurant entities: action, event** and **behaviour** entities, those
    - ⊗ that occur,
    - ⊗ that happen,
    - ⊗ that, in a sense, are accidents.



- **We think** that “decomposition” of the “data analysis” problem into
  - ❖ discrete parts and continuous materials,
  - ❖ atomic and composite parts,
  - ❖ their unique identifiers and mereology, and
  - ❖ their attributes
  - ❖ **is novel**,
  - ❖ and differs from past practices in domain analysis.

- In Sect. 11 we suggest
  - ◇ for each of the entity categories
    - ⊗ parts,
    - ⊗ materials,
    - ⊗ actions,
    - ⊗ events and
    - ⊗ behaviours,
  - ◇ a calculus of meta-functions:
    - ⊗ **analytic functions**,
      - \* that guide the **domain description developer**
      - \* in the process of selection,
      - and
    - ⊗ so-called **discovery functions**,
      - \* that guide that person
      - \* in “generating” appropriate **domain description texts**,  
informal and formal.

- The **domain description calculus** is to be thought of
  - ❖ as directives to the **domain engineer**,
  - ❖ mental aids that help a team of **domain engineers**
  - ❖ to steer it simply through the otherwise daunting task
  - ❖ of constructing a usually large **domain description**.
- Think of the **calculus**
  - ❖ as directing
  - ❖ a **human calculation**
  - ❖ of **domain descriptions**.
- Finally the **domain description calculus** section
  - ❖ suggests a number of **laws** that the
  - ❖ **domain description process** ought satisfy.

- Finally (Sect. 12) we bring a brief survey of the kind of requirements engineering
  - ⋄ that one can now pursue based on a reasonably comprehensive domain description.
  - ⋄ We show how one can systematically, but not automatically
  - ⋄ “derive” significant fragments
    - ⊗ of requirements prescriptions
    - ⊗ from domain descriptions.

- The formal descriptions will here be expressed in the **RAISE** [RaiseMethod] Specification Language, **RSL**.
- We otherwise refer to [TheSEBook1wo].
- Appendix brings a short primer, mostly on the syntactic aspects of **RSL**.
- But other **model-oriented formal specification languages** can be used with equal success; for example:
  - ❖ **Alloy** [alloy],
  - ❖ **Event B** [JRAbrial:TheBBooks],
  - ❖ **VDM** [e:db:Bj78bwo,e:db:Bj82b,jf-pgl-97] and
  - ❖ **Z** [m:z:jd+jcppw96].

## **End of Lecture 1: First Session — Introduction**

### **Domains, TripTych, Issues**

**FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012**



**SHORT BREAK**